Introduction to Computer Organization – Fall 2023

Homework 3

Due: @11:55pm, Monday November 6th on Gradescope

Name: Yuzhen Chen Uniqname: yuzhench

1. Submit a pdf of your typed or handwritten homework on Gradescope.

2. Your answers should be neat, clearly marked, and concise. Typed work is recommended, but not required unless otherwise stated. Show all your work where requested, and state any special or non-obvious assumptions you make.

3. You may discuss your solution methods with other students, but the solutions you submit must be your own.

4. **Late Homework Policy:** Submissions turned in by 1:00 am the next day will be accepted but with a 5% penalty. Assignments turned in between 1:00 am and 11:55 pm will get a 30% penalty, and any submissions made after this time will not be accepted.

5. When submitting your answers to Gradescope you need to indicate what page(s) each problem is on to receive credit. The grader may choose not to grade the homework if answer locations are not indicated.

6. **The last two questions are group questions**.
    - **You will turn those questions in separately** and may do it in a group of up to two students (yes, you can do it by yourself if you wish).
    - It is an honor code violation if a student is listed as contributing who did not actually participate in working on that problem.
    - Further, we suggest that you not split this up but rather work on the problem as a group.
7. After each question (or in some cases question part), we've indicated which lecture number we expect to cover the relevant material. So "**(L7)**" indicates that we expect to cover the material in lecture 7.

Problem 1: Intro to Pipelining (19 points) (L12)

For this problem consider the LC2K code segment below. Notice that there are no hazards.

Address				
0	add	3	2	3
1	lw	0	7	5
2	nor	2	4	1
3	sw	0	5	5
4	halt			
5	.fill	5		

a. Fill in the timing graph below with the pipeline stage that each instruction is in at the start of each cycle. Recall that the stages are IF, ID, EX, MEM, WB. The first IF has been filled in for you. **[4]**

Instruction \ Cycle #	0	1	2	3	4	5	6	7	8
add 3 2 3	IF	ID	EX	MEM	WB				
lw 0 7 5		IF	ID	EX	MEM	WB			
nor 2 4 1			IF	ID	EX	MEM	WB		
sw 0 5 5				IF	ID	EX	MEM	WB	
halt					IF	ID	EX	MEM	

b. Let the initial values of the registers be as follows:

Register	Value
0	0
1	-1
2	-3
3	9
4	2
5	0
6	-7
7	12

Fill in the contents of the **pipeline registers** at the point in time between cycle A and B. That is, “Cycle 3-4” means fill in the contents of the pipeline registers **after cycle 3 finishes, and before cycle 4 starts**. If a pipeline register is unused or you don’t care about the value, write an “X”. _Note that we’ve skipped some cycles, so cycle 1-2 doesn’t follow cycle 0-1, etc._ **[15]**

Example: **Cycle 0-1**

IF/ID		ID/EX		EX/MEM		MEM/WB	
Opcode:	add	Opcode:	noop	Opcode:	noop	Opcode:	noop
PC Plus 1:	1	PC Plus 1:	X	aluResult:	X	writeData:	X
		regA val:	X				
		regB val:	X	regB val:	X		
		offset:	X				

i. Cycle 2-3

IF/ID		ID/EX		EX/MEM		MEM/WB	
Opcode:	nor	Opcode:	lw	Opcode:	add	Opcode:	noop
PC Plus 1:	x	PC Plus 1:	x	aluResult:	6	writeData:	x
		regA val:	0				
		regB val:	x	regB val:	x		
		offset:	5				

ii. Cycle 4-5

IF/ID		ID/EX		EX/MEM		MEM/WB	
Opcode:	halt	Opcode:	sw	Opcode:	nor	Opcode:	lw
PC Plus 1:	x	PC Plus 1:	x	aluResult:	0	writeData:	5
		regA val:	0				
		regB val:	0	regB val:	x		
		offset:	5				

iii. Cycle 5-6

IF/ID		ID/EX		EX/MEM		MEM/WB	
Opcode:	x	Opcode:	halt	Opcode:	sw	Opcode:	nor
PC Plus 1:	x	PC Plus 1:	x	aluResult:	5	writeData:	0
		regA val:	x				
		regB val:	x	regB val:	0		
		offset:	x				

Problem 2: Data Hazards (20 points) (L14)
For this problem, reference the following piece of assembly code. All parts of this problem use the 5-stage LC2K pipeline datapath *as presented in lecture*. Assume all other memory locations are initialized to zero.

```
        lw      0       1       one             //lw1
        lw      0       2       num2            //lw2
        lw      1       3       num1            //lw3
        add     3       3       3               //add1
        lw      1       3       3               //lw4
        add     1       1       2               //add2
        nor     3       3       4               //nor1
        halt                                    //halt
one     .fill   1
num1    .fill   5
num2    .fill   3
```

a) Complete the table assuming **detect-and-stall** is used to deal with data hazards. Use ID* or IF* in the table to denote stalls in the decode or fetch stages . You may not need all the columns. **[10]**

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw1	IF	ID	EX	MEM	WB												
lw2		IF	ID	EX	MEM	WB											
lw3			IF	ID*	ID	EX	MEM	WB									
add1				IF*	IF	ID*	ID*	ID	EX	MEM	WB						
lw4						IF*	IF*	IF	ID	EX	MEM	WB					
add2									IF	ID	EX	MEM	WB				
nor1										IF	ID*	ID	EX	MEM	WB		
halt											IF*	IF	ID	EX	MEM		

b) Complete the table assuming **detect-and-forward** is used to deal with data hazards. Use Again, use ID* or F* in the table to denote stalls. You may not need all the columns. **[10]**

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw1	IF	ID	EX	MEM	WB												
lw2		IF	ID	EX	MEM	WB											
lw3			IF	ID	EX	MEM	WB										
add1				IF	ID*	ID	EX	MEM	WB								
lw4					IF*	IF	ID	EX	MEM	WB							
add2							IF	ID	EX	MEM	WB						
nor1								IF	ID	EX	MEM	WB.					
halt									IF	ID	EX	MEM					

Problem 3: CPI of Microarchitectures (6 points) (L15)

Say you have an LC2K program with the following characteristics:

- 30% of instructions are loads
- 10% are stores
- 20% are adds
- 15% are nors
- 25% are branches
- 30% of the instructions are dependent on the instruction immediately in front of them. You may assume that is true no matter what instructions are involved.
- 55% of branches are taken.

1) What would be the expected CPI of your program using detect-and-forward to resolve data dependencies and detect-and-stall for branches? Assume we are using the pipeline described in class. Clearly show your work. **[3]**

$CPI = 1 + \underbrace{0.3 \times 0.3}_{0.09} \times 1 + \underbrace{0.25 \times 3}_{0.75} = 1.84.$

normal   lw needs one stall   Beq, needs three stall.

2) What would be the expected CPI of your program using detect-and-forward to resolve data dependencies and predict-not-taken for branches? Assume we are using the pipeline described in class. Clearly show your work. **[3]**

$CPI = 1 + \underbrace{0.3 \times 0.3}_{0.09} \times 1 + \underbrace{0.25 \times 0.55 \times 3}_{0.4125} = 1.5025$

Normal.

All lw needs one stall if the right behind opcode depends on it.

55% Beq, go to the target Branch.

Problem 4: Cache Basics (5 points) (L16)

1) Say you have a cache as described in lecture 16[1] for a CPU with 32-bit addresses. Each block of the cache holds 32 bytes of data. How many bits are used for the tag? For the offset? **[2]**

Tag bits: $32 - 5 = 27$ Offset bits: $\log_2(32) = 5$

2) Describe, in your own words, why the "tag store" of the cache from cache is a CAM and the "data store" of a cache is not a CAM. **[3]**

The comparision is on the tag. and we need CAM to determine whether the decired tag is in the cash on not. But the data store is the position to store the actuall data value corresponding to the tags. And we only accessed those tags until they are needed. So this does not need a quick comparison.

[1] While we will define this term later, the cache described in lecture 16 is called a "fully associative" cache.
EDIT: Beaumont's lectures didn't cover this fully until L17

IF1 | IF2 | ID | EX | MEM1 | MEM2 | WB.

Problem 5: Putting it all together: Pipelines (Group, 25 points) (L15)
Your company has been producing a 1GHz version of the LC2K pipelined processor (as described in class, using detect-and-forward as well as predicting not-taken). You have been looking into designing a new, 1.75GHz version of the processor. But there is a problem--it turns out that everything can be sped up other than the memory. So the MEM and IF stages are going to be expanded into two parts each. So our new pipeline now has 7 stages (IF1, IF2, ID, EX, MEM1, MEM2, and WB). The "lw" and "sw" instructions will finish in MEM2. Branches and halt instructions also resolve in MEM2. Note: the question was updated on 11/4. We'll take answers that are consistent with either wording.

- 25% of instructions are loads
- 10% are stores
- 30% are adds
- 15% are nors
- 20% are branches
- ~~40% of the instructions are dependent on the instruction immediately in front of them.~~
- ~~20% of the instructions are NOT dependent on the instruction immediately in front of them but are dependent on the instruction before that.~~
- 40% of all loads are immediately followed by an instruction dependent on that load.
- 20% of all loads are immediately followed by an instruction not dependent on that load and the instruction after that instruction *is* dependent on the load.
- No instruction generates a value used by both the instruction immediately after it *and* the instruction two behind it.
- For the above two lines, you may assume that is true no matter what instructions are involved.
- 65% of branches are taken.

Thus we are comparing two processors:
- OLD: 1GHz, 5 stages
- NEW: 1.75GHz, 7 stages.

Answer the following questions:

A. Assuming there were no control or data hazards (i.e. CPI for both is 1.0), how much faster is the NEW processor than the OLD processor? Give this as a percentage[2]. **[2]**

$\frac{1.75}{1} = 1.75$ So 75% faster.

[2] There can be some confusion over how to use a percentage to represent the performance difference of two processors. In computer architecture, the exact way to use this is fairly well defined. In general, if we say Processor A is 50% of the speed of processor B, we mean A is half as fast. Which is the same as saying that A takes twice as long to do the task. If we say Processor A is 300% the speed of processor B, that means it is 3 times as fast.

You may think all of that is obvious, but people often will say things like "50% faster" to mean 150% the speed. And that leads to serious confusion. We often avoid percentages and say things like "a speedup of 2x" or a "speedup of 0.5x" (the second weirdly means it slowed down). See https://en.wikipedia.org/wiki/Speedup for more details.

B. What is the actual CPI for the OLD processor? **[3]**

$\boxed{CPI = 1 + 0.4 \times 0.25 \times 1 + 0.2 \times 0.65 \times 3} = 1.49$.

C. Explain *exactly* what the impact adding the two stages to the NEW processor has on control hazards. Be sure to explain exactly when this will occur and what the effect is. **[4]**

In the new design structure: there are two more instructions (IF2 & MEM1) before MEM2.

So when a branch misprediction occurs, we need to let two more pipe register to noop which means we have two more wasted cycles. So finally the CPI will increase.

D. Explain *exactly* what the impact adding the two stages to the NEW processor has on data hazards. Be sure to explain exactly when this will occur and what the effect is. **[4]**

For old pipline structure, if one instruction is directly behind "lw", then we neep one noop in order lew "lw" to get the correct data from datamemory. However, right now the new pipeline structure has two stage related to MEM (MEM1 & MEM2). So if one instruction directly behind "lw" and depends on destination register of "lw" then it needs two noops. And if one instruction is one gap behind "lw" and depends on destination register of "lw" then it needs one noops.

E. What is the CPI of the NEW processor? Show your work. **[6]**

$$\text{New CPI} = \underbrace{1}_{\text{normal}} + \underbrace{0.2 \times 0.25 \times 1 + 0.4 \times 0.25 \times 2}_{\text{lw (directly behind / one gap behind)}} + \underbrace{0.2 \times 0.65 \times 5}_{\text{beq (misprediction)}} = 1.9$$

F. Which processor is faster? How much faster (as a percentage as was done in part A). Show your work. **[6]**

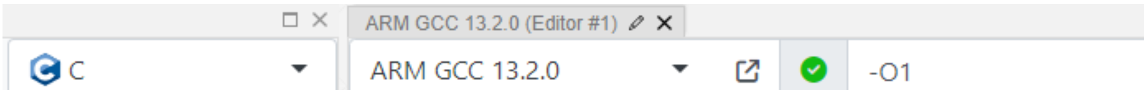
New: New CPI $\cdot$ 0.57 $= 1.9 \times 0.57 = 1.083$ ns/instruction.

old: old CPI $\cdot$ 1 $= 1.49$ ns/instruction.

The new processor is faster.

$1.49 \div 1.083 = 1.38$. So New processor is 38% faster than the old processor.

Problem 6: Looking at a real compiler with a slightly different ISA (Group, 25 points) (L6)
Go to https://godbolt.org/. Select the ARM GCC 13.2.0 compiler, the C programming language, and set the compiler options to "–O1" (turning on the optimizer). Note, this is a 32-bit version of ARM and among other differences, the registers are listed with an r rather than an X (so r1 rather than X1). Enter the following code and then answer questions in parts a through d. If you've set things correctly, the website bar should look something like this:




```
#include<stdio.h>
int fib(int n)
{
    int a,b;
    if (n <= 1)
        return n;
    a=fib(n-2);
    b=fib(n-1);
    return(a+b);
}

int main ()
{
    int n = 7;
    printf("%d",fib(n));
    return 0;
}
```

You might find https://developer.arm.com/documentation/ddi0597/2023-09/Base-Instructions?lang=en useful as a reference. Also, be sure to notice just how useful the colorization is.

a) Copy the assembly code for the function "fib". Comment each line of assembly for that function explaining what it does. You are going to have to look up some of the instructions online. We are not looking for things that just restate what the assembly instruction does (such as "stores register 2" or "copies register 3 to memory") but instead explains it in context ("puts the argument onto the stack" or "calls the function fib"). **[11]**

b) What will be the maximum stack depth of this program (in bytes)? Include main and briefly justify your answer. **[4]**

c) Explain how arguments are being passed, where the return value is being placed, and how caller/callee save issues are being resolved. **[4]**

d) Write a short C program which computes combinations recursively[3] and give it to the same compiler. Briefly provide the C code, the assembly code, and explain how arguments are passed. **[6]**

[3] The recursive definition to calculate n choose k can be defined as follows (for n, k non-negative integers):
C(n,k) = 1 If k = 0 or n = k;
Else C(n,k)=C(n-1, k) + C(n-1, k-1)